chương trình con

CHƯƠNG TRÌNH CON- THỦ TỤC VÀ HÀM

1. Khái niệm về chương trình con:

Chương trình con là một chương trình nằm bên trong một chương trình khác. Chương trình con có 2 loại: Thủ tục (Procedure) và hàm (Function):

- Thủ tục (PROCEDURE): Dùng để thực hiện một hay nhiều nhiệm vụ nào đó.
- Hàm (FUNCTION): Trả về một giá trị nào đó (có kiểu vô hướng, kiểu string hoặc kiểu con trỏ). Hàm có thể sử dụng trong các biểu thức.

Chương trình con được dùng rộng rãi khi xây dựng các chương trình lớn nhằm làm cho chương trình dễ theo dõi, dễ sửa chữa, có thể phân mảnh chương trình cho nhiều người làm. Một đặc điểm nổi bật của chương trình con là nó có tính đệ quy nhờ thế mà nhiều bài toán được giải quyết dễ dàng.

CẤU TRÚC CHUNG CỦA MỘT CHƯƠNG TRÌNH CÓ SỬ DỤNG CHƯƠNG TRÌNH CON:

```
 PROGRAM  Tên_chương_trình;

USES CRT;

CONST  ............;

TYPE    ............;

VAR     ............;



PROCEDURE  THUTUC[(Các tham số)];
```

```
[Khai báo Const, Type, Var]
BEGIN
    ..............
END;
 FUNCTION  HAM[(Các tham số)]:<Kiểu dữ liệu>;
[Khai báo Const, Type, Var]
BEGIN
    ..............
    HAM:=<Giá trị>;
END;

BEGIN {Chương trình chính}
    ...................
    THUTUC[(...)];
    ...................
    A:= HAM[(...)];
    ...................
END.
```

Chú ý: Trong quá trình xây dựng CHƯƠNG TRÌNH CON, khi nào thì nên dùng thủ tục/hàm?

Dùng hàm	Dùng thủ tục
- Kết quả của bài toán trả về 1 giá trị duy nhất (kiểu vô hướng, kiểu string hoặc kiểu con trỏ).- Lời gọi CHƯƠNG TRÌNH CON cần nằm trong các biểu thức tính toán.	- Kết quả của bài toán không trả về giá trị nào hoặc trả về nhiều giá trị hoặc trả về kiểu dữ liệu có cấu trúc (Array, Record, File).- Lời gọi CHƯƠNG TRÌNH CON không nằm trong các biểu thức tính toán.

(ví dụ n!, tìm điểm đối xứng)

Ví dụ 1.1: Viết CHƯƠNG TRÌNH CON để tính n! = 1.2...n.

```
Function GiaiThua(n:integer):integer;

Var P, i:integer;

Begin

   P:=1;

   For i:=1 To n Do P:=P*i;

   GiaiThua:=P;

End;
```

Ví dụ 1.2: Viết chương trình con để tìm điểm đối xứng của điểm (x,y) qua gốc tọa độ.

```
Procedure DoiXung(x,y:Integer; Var xx,yy:Integer);
Begin
    xx:=-x;
    yy:=-y;
End;
```

CHÚ Ý: Trong 2 ví dụ trên:

n, x, y được gọi là tham trị (không có từ khóa var đứng trước) vì sau khi ra khỏi chương trình con giá trị của nó không bị thay đổi.

xx, yy được gọi là tham biến (có từ khóa var đứng trước) vì sau khi ra khỏi chương trình con giá trị của nó bị thay đổi.

1. Tham số trong chương trình con:

Các chương trình con có thể không cần tham số mà chỉ có các biến riêng (biến cục bộ). Trong trường hợp cần nhận các giá trị mà chương trình mẹ truyền cho thì chương trình con cần phải có các tham số.

Tham số thực là những giá trị lưu trữ trong các biến toàn cục của chương trình mẹ, được truyền cho các thủ tục hoặc hàm thông qua lời gọi tên của chúng.

Tham số được khai báo ngay sau tên chương trình con được gọi là tham số hình thức. Tham số hình thức gồm:

- Tham biến:

Tham biến là những giá trị mà chương trình con nhận từ chương trình mẹ, các giá trị này có thể biến đổi trong chương trình con và khi chương

trình con kết thúc các giá trị này được trả về cho tham số thực.

Cách khai báo tham biến:

Tên chương trình con (Var tên tham biến: kiểu dữ liệu);

- Tham trị:

Tham trị là những tham số truyền vào cho chương trình con xử lý nhưng khi quay về chương trình mẹ vẫn phải giữ nguyên giá trị ban đầu.

Tên chương trình con (tên tham trị: kiểu dữ liệu);

1. Truyền tham số cho chương trình con:

Khi tham số hình thức trong chương trình con là tham biến thì tham số thực trong chương trình mẹ phải là biến chứ không thể là hằng. Trong mọi trường hợp cả hai tham số thực và tham số hình thức đều phải cùng kiểu dữ liệu…(các tham biến khi ra khỏi chương trình con giá trị thay đổi).

Khi tham số hình thức là tham trị thì tham số thực phải là một giá trị.

1. Biến toàn cục và biến địa phương:

- Biến toàn cục: là các biến được khai báo trong chương trình chính. Các biến này có tác dụng ở mọi nơi trong toàn bộ chương trình.
- Biến địa phương: là các biến được khai báo trong các chương trình con. Các biến này chỉ có tác dụng trong phạm vi chương trình con đó mà thôi.

Chú ý: Trong một chương trình con, nếu biến toàn cục trùng tên với biến địa phương thì biến địa phương được ưu tiên hơn.

Ví dụ 1.3:

```
Program  KhaoSatBien;

Var a,b: Integer;   {biến toàn cục}


Procedure  ThuBien;

Var  a: Integer; {biến địa phương}

Begin

    a:=10;

    Writeln(‘A=’,a,’B=’,b);

End;


Begin

    a:=50;

    b:=200;

    ThuBien;                         {A=10  B=200}

    Writeln(‘A=’,a,’B=’,b);      {A=50  B=200}

End.
```

1. Tính đệ quy của chương trình con:

Thông thường lời gọi một chương trình con chỉ được thực hiện khi chương trình con đó đã được thiết kế hoàn chỉnh. Tuy nhiên, Pascal còn cho phép một chương trình con ngay khi trong quá trình xây dựng lại có thể gọi tới chính nó, tính chất này được gọi là “Đệ quy của chương trình con”.

1. Lời gọi chương trình con:

Một chương trình mẹ có nhiều chương trình con trực thuộc, bên trong mỗi chương trình con lại có thể có các chương trình con riêng. Khi thiết kế, mỗi chương trình con phải là một khối riêng biệt hoặc có thể có các lệnh nhảy Goto từ chương trình con này tới chương trình con khác.

- Gọi chương trình con từ trong chương trình mẹ:

Lời gọi chương trình con có thể đặt bất kỳ chỗ nào trong chương trình mẹ. Nếu chương trình con là một thủ tục thì lời gọi chương trình con có thể tạo nên một câu lệnh, ví dụ:

Readln;

Nếu chương trình con là hàm thì tên hàm không thể tạo nên một câu lệnh, vì vậy tên hàm phải nằm trong một biểu thức hay trong một thủ tục nào đó. Ví dụ, ta không thể viết:

Sqrt(9);

gọi hàm như sau là hợp lệ: a:=sqrt(9)+5;

- Gọi chương trình con từ chương trình con khác:

Các chương trình con cùng cấp có thể gọi tới nhau và truyền tham số cho nhau. Nguyên tắc gọi là: những chương trình con xây dựng sau có thể gọi tới các chương trình con đã xây dựng trước nó, đồng thời các chương trình con cấp dưới cũng có thể gọi tới các chương trình con cấp trên nếu chúng cùng một gốc. Các chương trình con xây dựng trước muốn gọi tới các chương trình con xây dựng sau thì phải có chỉ báo forward.

Xét một số ví dụ sau:

Ví dụ 1.4

Program Goi_CTC;

```
Type dayso=array[1..60] of byte; S1:=string[30];

Var

a:s1; b:dayso; i,j,n:byte;

Procedure nhapso(m:byte; var c:dayso);

Begin

For i:=1 to m do

begin

Write(‘c[‘,i.’]=’); readln(c[i]);

End;

End;

Function tinhtong(m:byte; var d:dayso):real;

Var tong:real;

Begin

tong:=0;

For i:=1 to m do tong:=tong+d[i];

Tinhtong:=tong;

End;

Procedure Inkq(k:byte; e: dayso);

Begin
```

Write(‘tong cac ptu =’,tinhtong(k,e):8:0); {chương trình con gọi một chương trình con cùng cấp}

Readln;

End;

BEGIN

Write(‘nhap so ptu n’); readln(n);

Nhapso(n,b);

Inkq(n,b);

End.

Nếu hàm tinhtong xây dựng sau thủ tục Inkq, thì phải có chỉ báo forward.

Thêm dòng: Function tinhtong(m:byte; var d:dayso):real; forward; trước khi xây dựng các chương trình con.

Ví dụ 1.5

Program Goi_CTC;

Type dayso=array[1..60] of byte; S1:=string[30];

Var

a:s1; b:dayso; i,j,n:byte;

Procedure nhapso(m:byte; var c:dayso);

Begin

…

End;

```
Function tinhtong(m:byte; var d:dayso):real;
Var tong:real;
Begin
…
End;
Procedure xuly(j:byte;ds:dayso);
Procedure Inkq(k:byte;e:dayso);
Var i:byte;
Begin
Writeln('tong cac phan tu mang=',tinhtong(k,e):8:0;
Writeln ('day so sap xep giam dan');
For i:=1 to k do write(e[i],' ');
Readln;
End; {ket thuc thu tuc Inkq}
Procedure sapxep(m:byte;ds:dayso);
Var p,q:byte; tg:byte;
Begin
For p:=1 to m-1 do
For q:=p+1 to m do
If(d[p]<d[q] then
```

Begin

Tg:=d[p];

d[p]:=d[q];

d[q]:=tg;

end;

inkq(m,d); {goi den chuong trinh con cùng cấp}

end; {ket thuc thu tuc sap xep}

Begin {than thu tuc Xuly}

Write(‘thu tuc xu ly dung de sap xep va in ket qua}

Sapxep(j,ds);

End;

BEGIN

Write(‘nhap so phan tu’); readln(n);

Nhapso(n,b);

Xuly(n,b);

END.

Bài tập:

Tính $C_n^k = \frac{n!}{k!(n-k)!}$

(có sử dụng chương trình con)

Bản ghi

DỮ LIỆU KIỂU BẢN GHI

- Khái niệm cơ bản:

Bản ghi(Record) là một cấu trúc bao gồm một số (cố định hoặc thay đổi) các phần tử có kiểu khác nhau nhưng có liên quan nhau. Các phần tử này gọi là các trường (Field), vd: bảng điểm của lớp học bao gồm các trường Hoten, Ngaysinh, Toan,Ly, Hoa.. Dữ liệu điền vào các trường hình thành nên một bản ghi (Record). Có thể có những trường trong một bản ghi lại là một bản ghi, ví dụ trường Ngaysinh ở trên có thể là một bản ghi của 3 trường là Ngay, Thang, Nam. Bản ghi không phải là kiểu dữ liệu đã có sẵn trong Pascal mà do người sử dụng tự định nghĩa do đó chúng phải được khai báo ở phần TYPE.

Bản ghi bao gồm 2 loại:

- Bản ghi có cấu trúc không đổi
- Bản ghi có cấu trúc thay đổi

Điểm mạnh của bản ghi là cho phép xây dựng những cấu trúc dữ liệu đa dạng phục vụ công việc quản lý, tuy vậy muốn lưu trữ dữ liệu để sử dụng nhiều lần thì phải kết hợp kiểu bản ghi với kiểu Tệp.

- Khai báo:

```
TYPE    TênKiểu = RECORD
            Field1 : Kiểu1;
            Field2 : Kiểu2;
            ...
            FieldN: KiểuN;
        END;
```

VAR   Biến : TênKiểu;

Ví dụ 2.1:

TYPE   Bangdiem = RECORD

Hoten : String[25];

Gioitinh:Char;

Lop:String[5];

Diachi: String[30];

Toan,Ly,Hoa:Real;

END;

Tổng độ dài của Record là: 26+1+6+31+18=82 Bytes.

Có thể dùng hàm sizeof(tên kiểu) để xác định độ dài một kiểu dữ liệu, ví dụ:

Write(sizeof(bangdiem)) sẽ nhận được số 82.

Ví dụ 2.2

Xây dựng kiểu dữ liệu quản lý hồ sơ công chức:

Type

Diadanh=Record

Tinh,Huyen,Xa:String[15];

End;

Donvi=Record

Truong:String[30];

Khoa,Bomon:String[20];

End;

Ngay=Record

Ng:1..31;

Th:1..12;

Nam:Integer;

End;

Lylich=Record

Mhs:Word;

Hoten:String[25];

Ngaysinh:Ngay;

Quequan:Diadanh;

Coquan:Donvi;

End;

- Truy nhập các trường của bản ghi:

Giả sử cần quản lý danh sách cán bộ của một trường đại học, chúng ta phải khai báo một biến chứa danh sách:

Var DS:Lylich;

Để truy nhập các trường cần viết:

<tên biến>.<tên trường mẹ>.<tên trường con>…

Ví dụ để nhập dữ liệu cho trường Hoten ta viết các lệnh:

Readln(DS.hoten);

Để nhập ngày tháng năm sinh:

Readln(Ds.Ngay.Ng);

Readln(Ds.Ngay.Thang);

Readln(Ds.Ngay.Nam);

- Lệnh With…Do

Các cú pháp của lệnh:

WITH <Tên biến kiểu RECORD> DO <Các lệnh>

Khi sử dụng lệnh WITH..DO chuỗi lệnh viết sau DO chỉ cần viết tên trường có liên quan mà không cần viết tên biến.

Ví dụ 2.3

Program Nhapdiem;

Uses CRT;

Type

BANGDIEM=RECORD

Hoten:String[25];

Gioitinh: (‘1’,’0’);

Lop:String[5];

```
Toan,Ly,Hoa:Real;

End;

Var

DS_LOP:Array[1..40] of BANGDIEM {ds lớp là mảng 40 phần tử}

i,j: integer;

tiep:Char;

BEGIN

Clrscr;

tiep:=’C’; i:=1;

Repeat

With DS_LOP[i] do

Begin

Write (‘Ho ten hs:’); Readln(Hoten);

Write (‘Nam hay nữ: 1/0’); Readln(Gioitinh);

Write (‘Lop:’); Readln(Lop);

Write (‘Dia chi:’); Readln(Diachi);

Write (‘Diem Toan:’); Readln(Toan);

Write (‘Diem ly:’); Readln(Ly);

Write (‘Diem Hoa:’); Readln(Hoa);

End;
```

```
i:=i+1;

Write(‘Nhap tiep hay khong? C/K’); Readln(tiep);

Until upcase(tiep)=’K’;

Clrscr;

For j:=1 to i-1 do

With DS_LOP[j] do

Wrteln(Ho ten:15,’ ‘, Gioi tinh:2,’ ‘,Lop: 4,’ ‘,Dia chi:10,’ Toan:’, Toan:4:2,’ Ly:’, Ly:4:2, ’ Hoa:’, Hoa:4:2);

Readln;

END.
```

Đối với các bản ghi lồng nhau thì lệnh with..do cũng phải lồng nhau.

- Bản ghi có cấu trúc thay đổi:

a. Xây dựng kiểu dữ liệu:

Bản ghi có cấu trúc cố định dùng để mô tả một đối tượng mà các thể hiện của nó có các thuộc tính như nhau. Trong thực tế nhiều khi ta gặp những đối tượng mà thuộc tính của chúng gồm hai loại:

- Thuộc tính chung cho mọi thể hiện
- Thuộc tính riêng cho một số thể hiện đặc biệt

Ví dụ 2.4: Kiểu dữ liệu quản lý vé ngành đường sắt

Trên một tuyến đường sắt có nhiều đoàn tàu chạy trong ngày, có những chuyến tốc hành chỉ dừng lại ở một vài ga dọc đường, có những chuyến tàu thường dừng lại tất cả các ga lẻ. Với tàu tốc hành, hành khách chỉ được mang theo hành lý không quá 20 kg và sẽ có suất ăn trên tàu. Với tàu

thường hành khách phải mua vé hàng hóa nếu có vận chuyển hàng hóa và không có suất ăn trên tàu.

Thuộc tính chung: tên đoàn tàu (TDT), tuyến đường (TD), giờ đi (GD), loại tàu (LT) (ví dụ tốc hành –TH, tàu thường TT)

Thuộc tính riêng với tàu tốc hành: Số xuất ăn (SXA), số ga lẻ dừng dọc đường (SGD), còn tàu thường có thuộc tính riêng là cước hàng hóa (CHH).

Xây dựng các bản ghi dữ liệu:

Type QLDS=record

Ten_doan_tau:string[3];

Tuyen_duong:string[15];

Gio_di:real;

Case Loai_tau: (Toc_hanh,Tau_thuong) of

Toc_hanh:(So_xuat_an:Word; So_ga_do:Byte);

Tau_thuong: (cuoc_hang_hoa:real);

End;

Ví dụ 2.5: Kiểu dữ liệu quản lý điểm của sinh viên:

SV là kiểu dữ liệu bản ghi dùng để quản lý điểm của sinh viên. Các trường cố định của SV gồm MHS (mã hồ sơ), HOTEN (họ tên), NS (ngày sinh), GIOI (giới tính), Khoa (Sư phạm- SP, Kinh tế - KT, Cơ điện – CD). Các môn học tùy thuộc vào khoa mà sinh viên đang theo học, giả sử chúng ta quy định khoa sư phạm có các môn : Toán, Lý, Tin cơ bản, lập trình nâng cao; khoa Kinh tế có các môn : Kế toán máy, Marketing; khoa Cơ điện có các môn: Cơ học máy, Sức bền vật liệu. Tất cả sinh viên nếu

là Nam thì học thêm môn Bơi lội, nếu là Nữ thì học thêm Thể dục nghệ thuật.

Mỗi bản ghi kiểu sinh viên có cấu trúc thuộc một trong các dạng sau:

* Sinh viên khoa sư phạm:

1/ Mhs, Hoten, Ns, Boi_loi, Toan, Ly, Tincoban, Lap_trinh_nang_cao

2/ Mhs, Hoten, Ns, The_duc, Toan, Ly, Tincoban, Lap_trinh_nang_cao

* Sinh viên khoa cơ điện:

3/ Mhs, Hoten, Ns, Boi_loi, Co_hoc_may, Suc_ben_vat_lieu

4/ Mhs, Hoten, Ns, The_duc, Co_hoc_may, Suc_ben_vat_lieu

* Sinh viên khoa Kinh tế:

5/ Mhs, Hoten, Ns, Boi_loi, Ke_toan_may, Marketing

6/ Mhs, Hoten, Ns, The_duc, Ke_toan_may, Marketing

Kiểu dữ liệu bản ghi SV được khai báo như sau:

SV=record

Mhs: Byte;

Hoten: String[20];

NS:Record;

Ngay:1..31; Thang:1..12; Nam:Word;

End;

Case monhoc:(pl1,pl2) of

pl1: (case gioi:(Nam,Nu) of

Nam:(Boi loi:real);

Nu: (the_duc:real);

pl2: (case KHOA:(SP,CD,KT) of

SP: (Toan, Ly, Tincb, Ltnc:Real);

CD: (Co_hoc_may, Suc_ben_vat_lieu: real);

KT: (Ke_toan_may, Marketing:real));

End;

Từ cách khai báo trên, ta rút ra một số nhận xét sau:

Nhận xét 1: Trong một kiểu record các trường cố định được khai báo trước, trường phân loại khai báo sau, như vậy truòng phân loại phải là trường khai báo cuối cùng. Các trường thay đổi khai báo bên trong trường phân loại.

Nhận xét 2: Mỗi kiểu dữ liệu record có cấu trúc thay đổi chỉ được phép có duy nhất một trường phân loại, nghĩa là không thể có hai toán tử case…of ngang hàng khi khai báo.

Nhận xét 3: Vì mỗi trường lại có thể là một bản ghi cho nên bên trong trường phân loại lại có thể chứa các trường phân loại khác, đây là trường hợp bản ghi thay đổi nhiều mức.

b. Truy nhập:

Việc truy nhập vào các trường cố định của bản ghi có cấu trúc thay đổi hoàn toàn giống như bản ghi thường. Còn việc truy nhập vào các trường thay đổi cần phải chú ý một số điểm sau:

- Không dùng phép gán hoặc nhập dữ liệu từ bàn phím cho các trường phân loại.
- Lệnh With…do có tác dụng với tất cả các trường kể cả các trường thay đổi bên trong các trường phân loại.
- Tên các trường phân loại không thể đưa ra màn hình như một tên trường bình.

Bài tập chương 2:

1. Viết lại chương trình ở ví dụ 2.5, đưa vào chương trình con Nhập và chương trình con hiện. Dữ liệu đưa ra có dạng như sau:

Ma ho so	Ho va ten	Gioitinh	Khoa	So mon hoc	Tong diem	Trung bình
…						

Dữ liệu kiểu tệp

KIỂU DỮ LIỆU TỆP

1 Khái niệm về tệp:

Tệp là một dãy các phần tử cùng kiểu được sắp xếp một cách tuần tự. Tệp dữ liệu được lưu trữ ở bộ nhớ ngoài dưới một tên nào đó.

Tệp tập hợp trong nó một số phần tử dữ liệu có cùng cấu trúc giống như mảng nhưng khác mảng là số phần tử của tệp chưa được xác định.

Trong Pascal có 3 loại tệp được sử dụng là:

1. Tệp có kiểu:

Tệp có kiểu là tệp mà các phần tử của nó có cùng độ dài và cùng kiểu dữ liệu.

1. Tệp văn bản:

Dùng để lưu trữ dữ liệu dưới dạng các ký tự của bảng mã ASCII, các ký tự này được lưu thành từng dòng, độ dài các dòng có thể khác nhau. Ví dụ 2008 (kiểu word) khi ghi vào tệp văn bản cần 4 Byte (không phải 2 Byte).

1. Tệp không kiểu:

Tệp không kiểu là một loại tệp không cần quan tâm đến kiểu dữ liệu ghi trên tệp. Dữ liệu ghi vào tệp không cần chuyển đổi.

Tác dụng lớn nhất của kiểu dữ liệu tệp là ta có thể lưu trữ các dữ liệu nhập vào từ bàn phím và các kết quả xử lý trong bộ nhớ RAM ra tệp để dùng nhiều lần.

1. Khai báo:

- Định nghĩa kiểu tệp với từ khóa FILE OF trong phần mô tả kiểu sau từ TYPE, tiếp theo là khai báo biến tệp trong phần khai báo biến.

Ví dụ 2.6:

Type

MSN=Array[1..100] of integer; {định nghĩa mảng 100 số nguyên}

TSN= File of MSN; {định nghĩa tệp TSN có các phần tử là mảng số nguyên}

TCV=File of String[80]; {định nghĩa tệp TCV có các phần tử là các chuỗi có độ dài 80 ký tự.

Bangdiem= Record

……

End;

TBD= File of Bangdiem;

Var:

Tep1: TSN;

Tep2: TCV;

Tep3: TBD;

- Định nghĩa trực tiếp biến kiểu tệp trong phần khai báo biến

Var

Tep4:File of Array[1..5] of String[80];

Tep5: File of Bangdiem;

1. Truy nhập vào tệp:

Turbo Pascal có thể xử lý 2 loại tệp là : Tệp truy nhập tuần tự và tệp truy nhập trực tiếp.

- Tệp truy nhập tuần tự: để truy nhập vào một phần tử nào đó, ta bắt buộc phải đi qua các phần tử trước đó. Nếu muốn thêm các phần tử vào tệp thì có thể thêm vào cuối tệp.
- Tệp truy nhập trực tiếp: là tệp có thể truy nhập vào phần tử bất kỳ trong tệp. Muốn truy nhập trực tiếp phải dùng thủ tục Seek (số hiệu phần tử).
- Mở tệp:

Để mở một tệp chuẩn bị lưu trữ dữ liệu, ta sử dụng 2 thủ tục chuẩn sau đây:

ASSIGN(biến tệp, tên tệp);

REWRITE(biến tệp);

Trong đó:

Biến tệp: là tên biến tệp đã khai báo sau từ khóa VAR

Tên tệp: Là tên do ta chọn để ghi dữ liệu vào đĩa.

Ví dụ : ASSIGN(f, ‘a:\baitap.txt’);

REWRITE(f); {khởi tạo tệp rỗng}

Sau 2 thủ tục trên, để tiến hành ghi dữ liệu vào tệp ta lại dùng thủ tục WRITE(…):

Cách viết:

WRITE(biến tệp, các giá trị cần ghi vào tệp);

Cuối cùng, ta phải đóng tệp bằng thủ tục:

CLOSE(biến tệp);

2 Tệp văn bản:

a. Khai báo tệp văn bản:

Tệp văn bản được khai báo trực tiếp trong phần khai báo biến:

Var

Bientep:Text;

b. Truy nhập vào tệp:

Truy nhập vào tệp được hiểu là nhập dữ liệu vào tệp, ghi lại dữ liệu trên thiết bị nhớ ngoài, đọc dữ liệu đó ra màn hình hoặc máy in và xử lý nó.

- Mở tệp mới để ghi:

Assign(bientep, tentep);

Rewrite(bientep);

- Mở tệp đã có để ghi thêm:

Assign(bientep, tentep);

Append(bientep);

- Mở tệp để đọc dữ liệu:

Assign(bientep, tentep);

Reset(bientep);

c. Ghi dữ liệu vào tệp:

Sau khi đã mở tệp chúng ta có thể dùng thủ tục Write hoặc Writeln để ghi dữ liệu vào tệp.

Ví dụ 2.7:

```
Var
T1:Text;
Begin
Assign(T1,’Dulieu.dat’);
Rewrite(T1);
Writeln(T1,’Tep van ban’);
Write(T1,123);
Write(T1,’ ’,123.45);
Writeln(T1);
Close(T1);
End.
```

Dữ liệu ghi vào tệp như sau:

Tep van ban

123 1.234500000E+02

Dòng trống

d. Đọc dữ liệu từ tệp văn bản:

Sau khi tiến hành mở tệp, con trỏ tệp sẽ được đặt tại dòng đầu.Ta dùng thủ tục Read hoặc Readln để đọc dữ liệu từ dòng hiện thời và gán vào biến tương ứng, viết biến đó ra màn hình hoặc máy in.

Để có thể viết toàn bộ dữ liệu từ một tệp văn bản ra các thiết bị ngoài thì, thì các lệnh đọc viết phải được lặp đi lặp lại từ dòng 1 đến dòng cuối cùng, nghĩa là phải sử dụng một trong 2 vòng lặp:

While not eof(Bientep) do

Begin

Readln(Bientep, Dong); {biến Dong phải được khai báo trước, kiểu String}

Write(Dong);

End;

Hoặc:

For i:=1 to filesize(Bientep) do

Begin

Readln(Bientep,Dong);

Write(Dong);

End;

Lưu ý: Muốn lấy lại kiểu của dữ liệu nhập vào tệp văn bản thì mỗi biến phải nhập trên một dòng.

Ví dụ 2.8:

Xây dựng một chương trình đơn giản để quản lý công chức. Dữ liệu nhập bao gồm: Họ tên, Hệ số lương và số con. Dữ liệu xuất ra màn hình bao gồm Họ tên, Hệ số lương, Số con và Lương tháng (tính theo quy định của nhà nước = heso*540000).

Chương trình đặt ra hai khả năng lựa chọn:

1. Nếu tệp dữ liệu đã tồn tại thì nhập thêm người
2. Nếu tệp chưa có thì mở tệp mới

Trong cả 2 trường hợp đều cho biết số người cần nhập. Dữ liệu in ra dưới dạng bảng.

```
Program Quan_ly_can_bo;

Uses Crt;

Var f:Text; hoten:String[20]; c1, heso:real; c2,i,n,socon:byte;

Ten:string[12];

Begin

Clrscr;

Write(‘cho biet ten tep’); readln(ten);

Assign(f,ten);

Reset(f);

If IOResult=0 then

Append(f);

Else

Rewrite(f);

Write(‘nhap bao nhieu nguoi’); readln(n);

For i:=1 to n do

Begin

Write(‘Hoten’); Readln(hoten);
```

```
Write('He so'); Readln(heso);

Write('So con'); Readln(socon);

Writeln(f,hoten);

Writeln(f,heso:4:2);

Writeln(f,socon);

End;

Close(f);

Assign(f,ten);

Reset(f);

Writeln('-------------------------------------------------------');

Writeln ('| Ho va ten | Hs | socon | Luong |');

Writeln('-------------------------------------------------------')

While not eof(f) do

Begin

Readln(f,hoten);

Readln(f,heso);

Readln(f,socon);

Writeln('|', hoten:19,'|',heso:4:2,'|',socon:4,'|',heso*540000:10:2,'|');

End;

Readln;
```

End.

3 Tệp có kiểu:

a. Đọc và ghi :

- Ghi lên tệp:

Write(bientep,bien1,bien2,…);

bien1,bien2,…là các biến cùng kiểu với biến tệp.

- Đọc tệp:

Read(bientep,bien1,bien2,…);

Chú ý:

Khác với tệp văn bản, việc ghi và đọc tệp có kiểu không sử dụng các lệnh Writeln hoặc readln nghĩa là tệp có kiểu không ghi dữ liệu thành các dòng. Các phần tử của tệp có kiểu được ghi liên tục trong các ô nhớ và chỉ có ký hiệu kết thúc tệp EOF.

Khi chúng ta đọc hoặc ghi xong một phần tử thì con trỏ tệp sẽ tự động chuyển đến vị trí kế tiếp.

1. Truy nhập vào tệp:

Seek(bientep,i); i=0,1,2,…

Thủ tục seek sẽ định vị con trỏ tại vị trí thứ i của tệp.

1. các hàm xử lý tệp:

- Filesize(bientep) cho biết số phần tử có trong tệp
- FilePos(bientep) cho biết vị trí hiện thời của con trỏ tệp
- Eof(Bientep) cho giá trị là True nếu con trỏ tệp ở vị trí cuối tệp, ngược lại cho giá trị False

Ví dụ 2.9:

Tạo một tệp lấy tên là TEPCK.DAT để vừa ghi vừa sửa dữ liệu:

```
Program Tep_co_kieu:

Uses crt;

Var bt:file of byte; i:byte; n:real;

Begin

Clrscr;

Assign(bt,’ TEPCK.DAT’);

Rewrite(bt);

For i:=0 to 5 do write(bt,i); {ghi vào tệp 5 số nguyên}

Reset(bt);

Writeln(‘Du lieu luu tru trong tep TEPCK.DAT’);

While not eof(BT) do

Begin

Read(bt,i); write(i:5);

End;

Writeln;

Seek(bt,3); {định vị con trỏ tại phần tử thứ 4}

Textcolor(magenta);

Read(bt,i);
```

Writeln (‘So trong tep o vi trí thu 4:’,i);

i:=33;

seek(bt,3);

write(bt,i);

seek(bt,3); read(bt,i);

writeln(‘So moi trong tep o vi tri 4:,i);

writeln(‘vi tri hien thoi cua con tro:’, filepos(bt));

readln;

close(bt);

end.

4 Tệp không kiểu:

a. Khai báo biến tệp:

Var Bientep:File;

b. Mở tệp để ghi-đọc:

- Mở tệp mới để ghi:

Assign(bientep, tentep);

Rewrite(bientep, n);

- Mở tệp để đọc dữ liệu:

Assign(bientep, tentep);

Reset(bientep, n);

Với n là độ lớn tính theo Byte.

c. Đọc và ghi tệp không định kiểu:

* Đọc tệp không định kiểu:

BlockRead(bientep,biennho,i,j);

- biennho: là biến đã được khai báo cùng kiểu với các phần tử của tệp, biến nhớ đóng vai trò vùng nhớ đệm để lưu trữ dữ liệu đọc từ phần tử của tệp ra.
- i: là số phần tử quy định cho mỗi lần đọc.
- j: là biến kiểu Word, dùng để ghi lại số phần tử thực sự đã được đọc.

* Ghi tệp không định kiểu:

BlockWrite(bientep,biennho,i);

1. Truy nhập tệp không định kiểu:

Tệp không kiểu cũng được truy nhập như tệp có kiểu nghĩa là cũng dùng thủ tục Seek(bientep,n) để truy nhập vào phần tử thứ n+1 của tệp.

Lưu ý là với tệp không kiểu, mỗi lần con trỏ dịch chuyển nó sẽ dịch chuyển một số byte đúng bằng số byte đã quy định trong lệnh Rewrite() hoặc Reset()

Ví dụ 2.10

Nhập vào tệp các phần tử là record và sau đó viết chúng ra màn hình. Trong phần khai báo record chọn Hoten là string[15] và Diem thuộc kiểu Real.

Program tep_khong_kieu;

Uses Crt;

```
Type hs=record

Hoten:string[15];

Diem:real;

End;

Var

bt:file; k,nguoi:hs; i,j:byte;
        Begin
        Clrscr;
        assign(bt,'tep0kieu.dat');
        rewrite(bt,22);
        write('Nhap bao nhieu nguoi? ');
        readln(n);
          For i:= 1 to n  do
         with nguoi do
         Begin
         write('Ho va ten : '); readln(hoten);
         Write('Diem tong : '); readln(diem);
         blockwrite(bt,nguoi,1);
        end;
        for i:= 0 to n-1 do
        begin
        seek(bt,i);
        Blockread(bt,k,1);
        textcolor(red);
        with k do
        writeln(hoten,'  ',diem:5:2);
        end;

Close(bt);
```

```
Readln;

End.
```

Dữ liệu kiểu tập hợp

DỮ LIỆU KIỂU TẬP HỢP

1 Khái niệm tập hợp:

a. Khái niệm tập hợp:

Một tập hợp bao gồm n phần tử cùng kiểu dữ liệu (0<=n<=255), kiểu của các phần tử phải là kiểu vô hướng đếm được (nguyên, ký tự, logic, liệt kê, đoạn con). Số thứ tự các phần tử trong tập hợp luôn nằm trong khoảng 0-255.

b. Khai báo kiểu và gán dữ liệu vào tập hợp:

Để mô tả kiểu tập hợp, dùng từ khóa Set of tiếp đó là kiểu dữ liệu cơ bản của các phần tử tạo nên tập hợp.

Ví dụ:

Type

mau=set of (xanh, hong, tim, vang, den ,nau);

sn1:set of 0..100;

sn2: set of 100..200;

chucai1: set of ‘a’..’z’;

chucai2: set of ‘A’..’Z’;

Kytu=set of Char;

Var a:mau; b:sn1;

Khi khai báo kiểu tập hợp cần chú ý một số điều sau đây:

- Thứ tự các phần tử trong tập hợp luôn theo chiều tăng dần.

Nếu khai báo sn1=set of 200..100 -> sai

- Kiểu dữ liệu của các phần tử mặc dù có thể là số nguyên, nhưng khi khai báo là kiểu Byte.

Sn=set of Byte; -> đúng

Sn2 =set of Word; -> sai

- Những giá trị mà chúng ta khai báo sau từ khóa SET OF sẽ dẫn tới một trong 2 khả năng:

+ Nếu là khai báo kiểu liệt kê, kiểu Boolean thì phía sau từ khóa SET OF là giá trị mà tập hợp có thể nhận.

Ví dụ:

Nếu gán a:=[xanh,tim,luc]; -> sẽ bị báo lỗi.

+ Với những kiểu còn lại(nguyên, ký tự, đoạn con) những từ khai báo sau từ khóa Set of không phải là giá trị của các phần tử của tập hợp mà là thứ tự của phần tử đó. Kiểu của các dữ liệu đó sẽ cho biết các phần tử của tập hợp.

Type

sn1=set of 0..100;

kytu=set of char;

Var

a:sn1; b:kytu;

...

Với khai báo trên, ta không thể gán:

a:=['a',12,'c'];

b:=[1,2,100];

- Không được gán trực tiếp các số nguyên có giá trị lớn hơn 255

Vd b2:=[100,200,256]

2 Phân loại tập hợp:

a. Tập hợp cùng kiểu:

Các tập hợp được coi là cùng kiểu nếu chúng được xây dựng trên cùng một kiểu cơ bản.

Ví dụ:

Type

mau=set of (xanh,hong,tim,vang,den,nau);

mau_xe=set of (xanh..vang);

b. Hình thành một tập hợp:

Một tập hợp được hình thành khi ta gán cho biến tập hợp các giá trị cụ thể, các giá trị này được đặt trong hai dấu ngoặc vuông.

Ví dụ:

c:=[2,15,30,100..200];

chu:=[‘A’..’Z’];

3 Các phép tính trên tập hợp:

a. Phép gán:

Phép gán được dùng để hình thành nên một tập hợp. Có thể gán một tập hợp cho một biến tập hợp cùng kiểu hoặc gán một biến tập hợp cho cho một biến tập hợp khác cùng kiểu.

b. Phép hợp:

Hợp của 2 tập hợp A và B khi đó có thể viết là A+B

Hợp của 2 tập hợp A và B là một tập hợp gồm các phần tử thuộc A và các phần tử thuộc B.

Giả sử các tập hợp A,B,C được hình thành như sau:

A:=[1..8,10..15];

B:=[5..9,20,30];

C:=A+B;

- C gồm các phần tử [1..15,20,30];

c. Phép giao:

Giao của 2 tập hợp A và B, ký hiệu là A*B, là một tập hợp gồm các phần tử đồng thời thuộc A và thuộc B.

C:=A*B;

- C gồm các phần tử [5,6,7,8];

d. Phép trừ:

Phép trừ của 2 tập hợp A và B, ký hiệu là A-B, cho ta một tập hợp gồm các phần tử thuộc A và nhưng không thuộc B.

C:=A-B; C gồm các phần tử [1..4,10..15]

e. Phép thuộc về:

Phép thuộc về ký hiệu là IN là một phép thử xem một giá trị hay một biến có thuộc về một tập hợp hay không? Nếu có giá trị phép thử là True, ngược lại là False.

Ví dụ:

T1:=[‘a’..’z’];

T2:=’c’;

Write (T2 in T1);

Write(‘1’ in T1);

3.4 Các phép so sánh trên tập hợp:

a. Phép bằng, ký hiệu là =

Hai tập hợp A và B gọi là bằng nhau nếu mọi phần tử của A đều thuộc B và mọi phần tử của B đều thuộc A.

Ví dụ:

[1..6]=[1,2,3,4,5,6] cho kết quả là True

[‘a’,’b’,’c’]=[‘A’,’B’,’C’] cho kết quả False

b. Phép không bằng, ký hiệu <>

Hai tập hợp A và B gọi là không bằng nhau nếu tập A có ít nhất một phần tử không thuộc tập B hoặc tập B có ít nhất một phần tử không thuộc A.

[1..6]<>[1,2,3,4,5,6] cho kết quả là False

c. Phép nhỏ hơn hoặc bằng, ký hiệu <=:

Tập A gọi là nhỏ hơn hoặc bằng tập B nếu mọi phần tử của A đều thuộc tập B.

[1..6]<=[1,2,3,4,5,6,7] cho kết quả là True

d. Phép lớn hơn hoặc bằng, ký hiệu >=:

Tập A gọi là lớn hơn hoặc bằng tập B nếu mọi phần tử của B đều thuộc tập A.

[1..6]>=[1,2,3,4,5,6,7] cho kết quả là False

Bài tập 1: Trò chơi gieo xúc xắc: nguyên tắc chơi như sau: người chơi chọn một số trong khoảng từ 1 đến 6, máy sẽ gieo 3 con xúc xắc để được 3 số. Nếu số của người chơi nằm trong tập số mà máy đã gieo thì người chơi thắng, ngược lại người chơi thua. Nếu ba lần chơi mà người chơi đều thắng thì tuyên bố người chơi trúng số độc đắc.

```
Program xuc_xac;

Uses crt;

Var xucxac:set of 1..6;

So,i,dem,a,b,c:1..6; tl:char;

Begin

Clrscr;

Repeat

Clrscr;

For i:=1 to 3 do

Begin

Textcolor(5);

Write(‘moi ban chon mot so tu 1 den 6’); readln(so);

a:=random(6); b:=random(6); c:=random(6);

xucxac:=[a,b,c];
```

```
writeln('so ma ban da chon', so);

writeln('so may da gieo', a,' ',b,' ',c);

If so in xucxac then

Begin

Textcolor(4);

Writeln('Ban da thang van thu',i);

Dem:=dem+1;

End

Else

Writeln('Ban da thang van thu',i);

If dem=3 then Writeln('Ban trung doc dac – Xin chuc mung');

End;

Writeln;

Write('choi tiep hay k?C/K');

Readln(tl);

Until upcase(tl)='K';

End.
```

Bài tập 2: Viết chương trình nhập vào một chuỗi (chữ và số) , in ra kết quả số lần xuất hiện của mỗi chữ cái, in ra tổng số các chữ cái nhập từ bàn phím.

Đơn vị chương trình và thư viện chuẩn

ĐƠN VỊ CHƯƠNG TRÌNH VÀ THƯ VIỆN CHUẨN

1. Khái niệm đơn vị chương trình:

Thuật ngữ Unit trong Pascal được gọi là :”Đơn vị chương trình”. Mỗi Unit được xem như một Modul nhỏ chứa đựng một số công cụ cần thiết giúp cho người lập trình dễ dàng thiết kế chương trình.

Lệnh tham chiếu đến Unit được đặt ở đầu chương trình với cú pháp:

USES tênUnit;

Ví dụ: Uses CRT,Graph;

Các Unit được tổ chức trong Pascal dưới 2 dạng:

* Các file độc lập với phần mở rộng là TPU(Turbo Pascal Unit), ví dụ Graph.TPU

* File thư viện chuẩn với phần mở rộng TPL, ví dụ Turbo.TPL

Khi file Turbo.exe được gọi, nghĩa là chương trình Pascal được khởi động từ file Turbo.tpl cũng tự động được tải vào bộ nhớ. Lúc này các Unit chứa trong các thư viện chuẩn sẽ sẵn sàng được tham chiếu đến. Việc truy cập đến các Unit trong thư viện chuẩn nhanh hơn so với truy cập vào các Unit độc lập vì chúng đã thường trú trong bộ nhớ.

2. Thư viện chuẩn:

Thư viện chuẩn của Pascal có tên là Turbo.tpl, thư viện này chứa 5 Unit:

2.1 Crt

CRT bao gồm các thủ tục quản lý màn hình, bàn phím và âm thanh.

2.2 Dos:

Unit này chứa các chức năng quản lý tệp, đĩa, ngày tháng.

2.3 Overlay:

Unit này được sử dụng khi chương trình nguồn có dung lượng lớn. Sử dung Overlay để tải từng phần chương trình nguồn vào bộ nhớ để chạy.

2.4 Printer:

Unit này định nghĩa tên máy in là LST. Việc kết xuất thông tin bằng lệnh Write khi tham chiếu đến LST sẽ cho phép ta in ra máy in các kết quả bài toán.

2.5 System:

Đây là đơn vị cơ bản và quan trọng nhất của pascal, nó chứa các thủ tục vào ra như Read, Write…Khi Pascal được khởi động và thư viện chuẩn được nạp vào bộ nhớ thì Unit này cũng tự động liên kết với mọi chương trình, vì thế ở đầu chương trình không cần đến lời gọi Uses System.

1. Xây dựng Unit:

Cấu trúc một Unit bao gồm 4 phần cơ bản sau:

- Phần tiêu đề:

Unit tenUnit;

Lưu ý rằng Pascal quy định bắt buộc tên Unit sau này sẽ được dùng làm tên tệp để lưu trữ tệp nguồn của Unit.

- Phần khai báo chung

Bắt đầu bằng từ khóa INTERFACE.

Trong phần này ta khai báo các kiểu dữ liệu mới, các biến, hằng, hàm, thủ tục mà sau này các chương trình tham chiếu đến các Unit sẽ sử dụng.

Ví dụ chúng ta xây dựng một Unit lấy tên là HHP (hình học phẳng) trong đó có các hàm tính diện tích, chu vi các hình chữ nhật, tam giác, hình tròn. Khi đó phần khai báo chung sẽ là:

INTERFACE

Function dtcn(a,b:real):real;

Function dttg(a,b,c:real):real;

Function dttr(a:real):real;

Function cvcn(a,b:real):real;

Function cvtg(a,b,c:real):real;

Function cvtr(a:real):real;

- Phần nội dung:

Bắt đầu bằng từ khóa IMPLEMENTTATION. Tại đây ta sẽ xây dựng các hàm, thủ tục mà tên của chúng đã được giới thiệu ở phần Interface.

Ví dụ:

IMPLEMENTATION

Function dtcn(a,b:real):real;

Var s:real;

Begin

S:=a*b;

Dtcn:=s;

End;

…..

Function cvcn(a,b:real):real;

Var c:real;

Begin

c:=(a+b)*2;

cvcn:=c;

End;

….

- Phần khởi động:

Phần khởi động đặt giữa 2 từ khóa Begin và End, sau End là dấu . Trong phần này ta đưa vào các lệnh gán giá trị ban đầu cho các biến.

Phần khởi động không bắt buộc phải có, trong trường hợp không có phần này thì ta bỏ đi từ khóa Begin.

* Cấu trúc tổng thể của một Unit:

UNIT <Tên Unit>; {phải trùng với tên file}

INTERFACE

USES ....;

CONST.........;

TYPE ....;

VAR ....;

Procedure <Tên thủ tục>[(Các tham số)];

Function <Tên hàm>[(Các tham số)]:<Kiểu hàm>;

```
IMPLEMENTATION
    Procedure  <Tên thủ tục>[(Các tham số)];
    [Các khai báo]
    Begin
        .............
    End;


    Function  <Tên hàm>[(Các tham số)]:<Kiểu hàm>;
    [Các khai báo]
    Begin
        .............
    End;
END.
```

- Phần hướng dẫn:

Mỗi khi xây dựng xong cần có phần hướng dẫn để người sử dụng không gặp phải các lỗi khi chương trình tham chiếu đến Unit

Ví dụ: cách sử dụng Unit HHP tính diện tích chu vi tam giác

```
Program tinh_dtcv;
Uses crt,hhp;
Var m,n,q:real;
Begin
```

```
Clrscr;

Write('Cho biet 3 canh'); readln(a,b,c);

If(a+b>c) and (a+c>b) and (b+c>a) then

Begin

Writeln( 'Dien tich tam giac la',dttg(a,b,c):5:2);

Writeln( 'chu vi tam giac la',cvtg(a,b,c):5:2);

End

Else

Writeln('So lieu da cho khong tao thanh tam giac');

Readln;

End.
```

Bài tập 1 a) Xây dựng Unit HHP. Unit này tạo nên một số hàm dùng để tính diện tích, chu vi các hình tròn, chữ nhật, tam giác.

b) sử dụng Unit HHP tính diện tích, chu vi hình CN, hình tròn.

Bài tập 2: a) Tạo Unit MYTOOL lưu ở file MYTOOL.PAS.

```
UNIT  MYTOOL;

INTERFACE

   USES CRT;

   VAR  m:Integer;

   Function  UCLN(a,b:Integer):Integer;

   Function  NGUYENTO(n:Word):Boolean;
```

```
IMPLEMENTATION
    Function  UCLN(a,b:Integer):Integer;
    Begin
        While a<>b Do
           Begin
               If a>b Then a:=a-b  Else  b:=b-a;
           End;
        UCLN:=a;
    End;
    Function  NGUYENTO(n:Word):Boolean;
    Var d,i:Word;
    Begin
        d:=0;
        For i:=2 To n DIV 2 Do
           If n MOD i=0 Then d:=d+1;
        NGUYENTO:=d=0;
    End;
END.
```

b) Viết một chương trình có sử dụng Unit MYTOOL.

```
Uses Crt, MyTool;
Var  a,b:Integer;
```

```
Begin
    CLRSCR;
    Write(10,5,'CHUONG TRINH MINH HOA');
    Write('Nhap a = '); Readln(a);
    Write('Nhap b = '); Readln(b);
    Writeln('UCLN cua ',a,' va ',b,' la:',UCLN(a,b));
    Write('Nhap m = '); Readln(m);
    If  NGUYENTO(m) Then
        Writeln(m,' la so nguyen to!')
    Else
        Writeln(m,' khong phai la so nguyen to!');
    Readln;
End.
```

4. Một số Unit chuẩn:

4.1 System:

Đây là Unit cơ bản và quan trọng nhất của Pascal, nó chứa các thủ tục vào ra như Read, Write…, các hàm sơ cấp thông dụng như Ln,Sqrt, Sin, Cos… Khi Pascal được khởi động và thư viện chuẩn được nạp vào bộ nhớ thì Unit này cũng tự động liên kết với mọi chương trình, vì thế ở đầu chương trình không cần đến lời gọi USES SYSTEM.

4.2 Dos:

Unit này chứa các chức năng quản lý tệp, đĩa, ngày tháng.Ngoài ra, có thể dùng Unit này để gọi trực tiếp các lệnh của HĐH Dos.

4.3 CRT:

CRT là Unit liên quan đến các thủ tục trình bày màn hình. Trong CRT có các thủ tục sau:

1. Gotoxy(m,n):

Chuyển con trỏ tới tọa độ cột m và dòng n. (1<=m<80; 1<=n<=25)

1. CLRSCR: xóa màn hình
2. Delay(n): làm chậm lệnh phía trước n mili giây. Giá trị lớn nhất là 65536 ms.
3. Thuộc tính màu

Có 2 thủ tục gán thuộc tính màu cho ký tự viết ra màn hình là:

TEXTCOLOR(Mã màu): màu chữ

TEXTBACKGROUND(Mã màu) : màu nền

Bảng 3.1

Mã màu	Tên hằng	Màu	Mã màu	Tên hằng	Màu
0	Black	Đen	4	Red	Đỏ
1	Blue	Xanh lam	5	Magenta	Tím
2	Green	Xanh lá cây	6	Brown	Nâu

3	Cyan	Xanh lơ	7	LightGray	Xám nhạt
8	DackGray	Xám sẫm	12	LightRed	Đỏ sáng
9	LightBlue	Xanh lam sáng	13	LightMagenta	Tím sáng
10	LightGreen	Xanh lá mạ sáng	14	Yellow	Vàng
11	Lightcyan	Xanh lơ sáng	15	White	Trắng

1. Hàm keypressed:

Hàm keypressed kiểm tra xem trong quá trình làm việc có phím nào được bấm hay không?

Vd:

If keypressed then…

Hoặc Repeat … until keypressed;

1. Hàm Readkey:

Hàm Readkey nhận diện phím được bấm, giá trị mà nó nhận về là một ký tự

1. Hàm Getkey:

Hoạt động như hàm Readkey nhưng giá trị mà nó nhận là một số dương đối với các phím thông thường, nếu là các phím có mã quét mở thì giá trị Getkey nhận gồm 2 số: số đầu là số 0 và số thứ 2 là mã quét mở của phím.

Ví dụ: nếu bấm J thì Getkey cho số 75. Nếu nhấn <- thì Getkey cho 0-75

Enter: 13

Esc:27

Bài tập mẫu: Thiết kế menu với 4 tùy chọn:

Tamgiac chunhat Tron Ketthuc

Để chuyển con trỏ đến một chức năng nào đó ta bấm phím <- hoặc->, để chọn chức năng, ta bấm phím Enter, kết thúc công việc ấn phím End.

```
Program Menu;

Uses CRt;

label h1,h2,h3;

Var c1,c2,c3,c4:String[20];

l1,l2,l3,l4:byte;

chon:char;

Procedure tr; {thu tuc tinh hinh tron}

Var a,dt,cv:real;

Begin

clrscr;

write('cho biet ban kinh hinh tron'); readln(a);

dt:=pi*a*a; cv:=2*pi*a;
```

```
writeln('dien tich hinh la:',dt:12:4);

writeln('chu vi hinh la:',cv:12:4);

writeln('bam enter quay ve menu');

repeat until Keypressed;

end;

Procedure cn; {thu tuc tinh hinh chu nhat}

Var a,b,p,dt,cv:real;

Begin

clrscr;

write('cho biet do dai 2 canh'); readln(a,b);

dt:=a*b; cv:=(a+b)/2;

writeln('dien tich hinh la:',dt:12:4);

writeln('chu vi hinh la:',cv:12:4);

writeln('bam enter quay ve menu');

repeat until Keypressed;

end;

Procedure tg; {thu tuc tinh tam giac}

Var a,b,c,p,dt,cv:real;

Begin

clrscr;

write('cho biet ba canh tam giac'); readln(a,b,c);
```

```
if (a+b>c) and (a+c>b) and (b+c>a) then
begin
p:=(a+b+c)/2;
dt:=sqrt(p*(p-a)*(p-b)*(p-c));
cv:=a+b+c;
writeln('dien tich hinh la:',dt:12:4);
writeln('chu vi hinh la:',cv:12:4);
end
else
write('so lieu nhap khong tao thanh tam giac');
writeln('bam enter quay ve menu');
repeat until Keypressed;
end;
Procedure w1; {thiet ke chuc nang thu nhat}
Begin
Textbackground(5);
textcolor(10);
window(1,2,l1+1,2);write(c1);
gotoxy(1,1);
End;
Procedure w2;
```

```
Begin
Textbackground(5);
textcolor(10);
window(l1+2,2,l1+l2+2,2);write(c2);
gotoxy(1,1);
End;
Procedure w3;
Begin
Textbackground(5);
textcolor(10);
window(l1+l2+3,2,l1+l2+l3+3,2);write(c3);
gotoxy(1,1);
End;
Procedure w4;
Begin
Textbackground(5);
textcolor(10);
window(l1+l2+l3+4,2,l1+l2+l3+l4+4,2);write(c4);
gotoxy(1,1);
End;
Begin{THAN CHUONG TRINH CHINH}
```

```
Clrscr;
c1:='Tamgiac';l1:=length(c1);
c2:='Chunhat';l2:=length(c2);
c3:='Tron';l3:=length(c3);
c4:='Ketthuc';l4:=length(c4);
CLRSCR;
textcolor(red);
textbackground(green);
write('bam ->,<- di chuyen menu| Bam enter de chon| Bam End ket thuc');
h1: w1;w2;w3;w4;w1;
while keypressed do chon:=readkey;
chon:=readkey;{tinh tam giac}
if ord(chon)=13 then
begin
textcolor(blue);textbackground(14);
window(1,4,80,25);
tg;
clrscr;
goto h1;
end
else
```

```
begin
if ord(chon)=0 then chon:=readkey;
if ord(chon)=79 then halt; {bam End ket thuc}
if ord(chon)=77 then w2;
end;
h2: w1;w2;w3;w4;w2;
while keypressed do chon:=readkey;
chon:=readkey;{tinh chu nhat}
if ord(chon)=13 then
begin
textcolor(blue);textbackground(14);
window(1,4,80,25);
cn;
clrscr;
goto h2;
end
else
begin
if ord(chon)=0 then
begin
chon:=readkey;
```

```
if ord(chon)=79 then halt; {bam End ket thuc}

if ord(chon)=77 then w3;

if ord(chon)=75 then goto h1;

end;

end;

h3: w1;w2;w3;w4;w3;

while keypressed do chon:=readkey;

chon:=readkey; {tinh hinh tron}

if ord(chon)=13 then

begin

textcolor(blue);textbackground(14);

window(1,4,80,25);

tr;

clrscr;

goto h3;

end

else

begin

if ord(chon)=0 then

begin

chon:=readkey;
```

```
if ord(chon)=79 then halt; {bam End ket thuc}

if ord(chon)=77 then w4;

if ord(chon)=75 then goto h2;

end;

end;

while keypressed do chon:=readkey;

chon:=readkey; {tinh hinh tron}

if ord(chon)=13 then halt

else

if ord(chon)=0 then

chon:=readkey;

if ord(chon)=79 then halt; {bam End ket thuc}

if ord(chon)=75 then goto h3;

END.
```

Bài tập: Lập chương trình tạo menu 2 mức, mức 1 theo chiều ngang, mức 2 theo chiều dọc theo mẫu:

HINH PHANG	HINH KHONG GIAN
Tam giac	Hinh tru

Tron	Hinh cau

Yêu cầu: khi con trỏ ở một trong 2 chức năng HINH PHANG, HINH KHONG GIAN, nếu bấm Enter thì xuất hiện menu phía dưới, nếu bấm phím mũi tên thì sẽ chuyển con trỏ sang phải hoặc trái. Khi menu mức 2 xuất hiện sẽ có con trỏ dich chuyển lên xuống, bấm Enter thực hiện tính diện tích, chu vi (đối với hình phẳng), hoặc thể tích đối với Hình không gian).

Con trỏ và cấu trúc động

CONTRỎ VÀ CẤU TRÚC ĐỘNG

1. Khái niệm:

Khi khai báo một biến, dù là biến đơn hay biến thuộc kiểu dữ liệu có cấu trúc, ta đã quy định độ lớn vùng nhớ dành cho biến:

VD: a: real; biến a cần 6 byte

B: array[1..100] of integer; biến mảng b cần 200 byte

Việc khai báo như trên thường là phỏng đoán dung lượng cần thiết chứ không thật chính xác, gây nên lãng phí bộ nhớ.

Để tiết kiệm bộ nhớ, ngay khi chương trình đang làm việc, người lập trình có thể yêu cầu cấp phát bộ nhớ cho các biến, điều này gọi là cấp phát bộ nhớ động. Cấp phát bộ nhớ động được thực hiện thông qua biến con trỏ. Muốn có biến con trỏ ta phải định nghĩa kiểu con trỏ.

Kiểu dữ liệu con trỏ-biến con trỏ:

- Con trỏ có định kiểu:

Kiểu con trỏ là một kiểu dữ liệu đặc biệt dùng để biểu diễn các địa chỉ. Kiểu con trỏ được định nghĩa theo cú pháp:

Tên kiểu con trỏ=^Kiểu dữ liệu;

Ví dụ 4.1:

Chu=String[20];

CT1=^Byte;

CT2=^Chu;

CT3=^Nguoi;

Nguoi=record

Hoten:String[20];

Namsinh:1900..2100;

End;

Chú ý: ta chỉ được phép đưa trực tiếp vào định nghĩa kiểu con trỏ các kiểu dữ liệu đơn giản sau: số nguyên, số thực, ký tự. Các kiểu dữ liệu có cấu trúc muốn đưa vào con trỏ thì phải thông qua một tên kiểu khai báo trong phần Type.

Cách định nghĩa 2 kiểu con trỏ Hoten và Ds sau là sai:

Type

Hoten=^String[20];

Ds=Array[1..10] of Byte;

Muốn sử dụng kiểu chuỗi và mảng cho kiểu con trỏ chúng ta phải định nghĩa như sau:

Type

S1=String[20];

Hoten=^S1;

a=array[1..10] of Byte;

Ds=^a;

- Biến con trỏ:

Biến con trỏ có thể khai báo thông qua kiểu con trỏ hoặc khai báo trực tiếp.

Ví dụ 4.2:

Var

So:^integer;

Sinhvien:CT3;

Hoten:CT2;

Thutu, Mahoso:^Word;

Biến con trỏ không dùng để lưu trữ các giá trị của biến mà lưu trữ địa chỉ của biến. Dù kích thước vùng dữ liệu mà các biến con trỏ trỏ tới khác nhau nhưng kích thước của biến con trỏ vẫn là 4 byte.

- Con trỏ không định kiểu:

Là kiểu con trỏ không quan tâm đến kiểu dữ liệu mà nó trỏ tới.

Cách khai báo:

Var tên biến: Pointer;

- Địa chỉ của một đối tượng:

Địa chỉ một đối tượng trong bộ nhớ được xác định bởi địa chỉ của ô nhớ đầu tiên mà hệ thống dành cho đối tượng đó.

$0101	$FFFF

Segment (địa chỉ đoạn)	Offset(địa chỉ tương đối trong đọan)

Địa chỉ ô thứ 65535, thuộc đoạn 257.

Các thủ tục và hàm tác động lên con trỏ:

- Gán giá trị ban đầu:

Ct:=nil;

- Gán địa chỉ của một đối tượng cho con trỏ:

1. ct:=@x;

b. ct:=Addr(x)

Hàm Addr()cho địa chỉ của đối tượng x, địa chỉ này thuộc kiểu Pointer

c. ct:=Ptr(segment):

Hàm Ptr trong phép gán trên đòi hỏi các tham số segment và offset phải là giá trị kiểu Word viết trong hệ 16, ví dụ :

ct:= Ptr($B800,$0000);đưa con trỏ trỏ tới ô nhớ của vùng Video Ram

Nhận xét:

Hai phép gán @ và Addr() cùng trả về địa chỉ kiểu pointer nên chúng là tương đương.

3. 3 phép gán giữa hai con trỏ

Hai con trỏ tương thích (cùng kiểu) có thể gán giá trị cho nhau, khi đó chúng cùng trỏ tới một địa chỉ .

Ví dụ 4.3

Var

ct1:^Float;

ct2:^Byte;

ct3:^Pointer;

x:string;

Ví dụ trên khai báo ba con trỏ thuộc ba kiểu khác nhau, ct1 là con trỏ thực,ct2 là con trỏ nguyên và ct3là con trỏ không định kiểu, x là biến chuỗi. Khi đó các phép gán :

ct3:@x;

ct2=ct3;

là hợp lệ vì ct2 và ct3 là tương thích, chúng cùng trỏ đến địa chỉ cùng biến x

ct1:=ct2;

là không hợp lệ vì hai con trỏ không tương thích .

3. 4 Phép so sánh hai con trỏ

Chỉ tồn tại phép so sánh =(bằng nhau)và<>(khác nhau)giữa hai con trỏ nếu chúng tương thích. Kết quả so sánh là một giá trị Boolean nghĩa là True hoặc False.

Hai con trỏ tương thíchgọi là bằng nhau nếu chúng cùng trỏ tới một đối tượng , ngược lại gọi là khác nhau.

4.Truy nhập dữ liệu

Khi con trỏ ct đang trỏ tới một vùng dữ liệu nào đó Pascal cho phép dùng ký hiệu ct^ như là một biến để truy nhập vào vùng dữ liệu đó . Biến ct^ mang trong đó dữ liệu của vùng mà con trỏ ct đang trỏ tới.

Như vậy chúng ta có thể truy nhập tới một biến, hàm hai thủ tục mà không cần biết tên các đối tượng này miễn là biết con trỏ đang trỏ vào chúng .

Ví dụ 4.5

```
Program contro;

Uses crt;

Type zl= string[3];

Var

z:string;ct:^zl;i:byte;

Begin

clrscr;

z:='Ha noi';ct:=@z;

writeln(ct^);

For i:=1 to length(z) do write(upcase(ct^[i]);

Readln;

End.
```

Chạy chương trình ta nhận được kết quả:

Ha noi

HA NOI

Mọi xử lý trên biến z đều có thể xử lý trên biến ct^ bởi vì biến con trỏ ct đang trỏ vào z.

1. Mảng con trỏ và con trỏ kiểu mảng:

Con trỏ là một kiểu dữ liệu nên biến con trỏ có thể là các thành phần của mảng, ngược lại mảng là một kiểu dữ liệu có cấu trúc nên con trỏ cũng có thể trỏ tới các biến mảng.

5.1 Con trỏ kiểu mảng:

Khai báo:

Type m= array[1..5] of Byte;

Var

Ct1:^m;

Ct1 là biến con trỏ kiểu mảng, khi đó biến ct1^ sẽ gồm 5 phần tử, mỗi phần tử là một số kiểu Byte.

Truy cập vào biến ct1^:

Read(ct1^[i]); hoặc Write(ct1^[i]);

5.2 Mảng các con trỏ:

Khai báo:

Var:

Ct:array[1..10] of ^string;

s1,s2:String;

Begin

s1,s2:String;

Begin

S1:=’Ha noi Viet nam’;

S2:=’Happy new Year’;

…

Ct là mảng 10 con trỏ, tất cả 10 con trỏ trỏ tới đến kiểu dữ liệu String. Mỗi con trỏ trỏ đến một đối tương khác nhau.

- Nếu ta chưa gán địa chỉ của bất kỳ đối tượng nào cho biến con trỏ mà chỉ thực hiện phép gán:

Ct[i]:^=s1; với 1<=i<=10

Thì 10 con trỏ đều trỏ tới s1;

- Trong trường hợp ta gán dữ liệu từ một đối tượng cho nhiều biến con trỏ thì tất cả các con trỏ trỏ tới đối tượng được gán cuối cùng.

Nếu thực hiện phép gán:

Ct[1]:=@s2;

Nghĩa là gán địa chỉ của biến s2 vào con trỏ thứ nhất trong mảng thì chỉ có ct[1] là trỏ tới biến s2, các con trỏ con lại chưa trỏ vào đâu cả.

1. Danh sách liên kết và hàng đợi:

Danh sách là một tập hợp hữu hạn các phần tử liên kết với nhau, trường hợp tổng quát nhất mỗi phần tử là một bản ghi. Điều đặt biệt của mỗi bản ghi trong danh sách là ngoài các trường dữ liệu, còn một trường dùng để liên kết và trường này lại là một con trỏ. Con trỏ này có nhiệm vụ trỏ vào địa chỉ của bản ghi kế tiếp. Nếu bản ghi hiện thời là bản ghi cuối cùng thì con trỏ sẽ trỏ vào Nil.

Một danh sách chưa có phần tử nào được gọi là danh sách rỗng. Việc thêm một phần tử vào danh sách có thể rơi vào một trong ba khả năng:

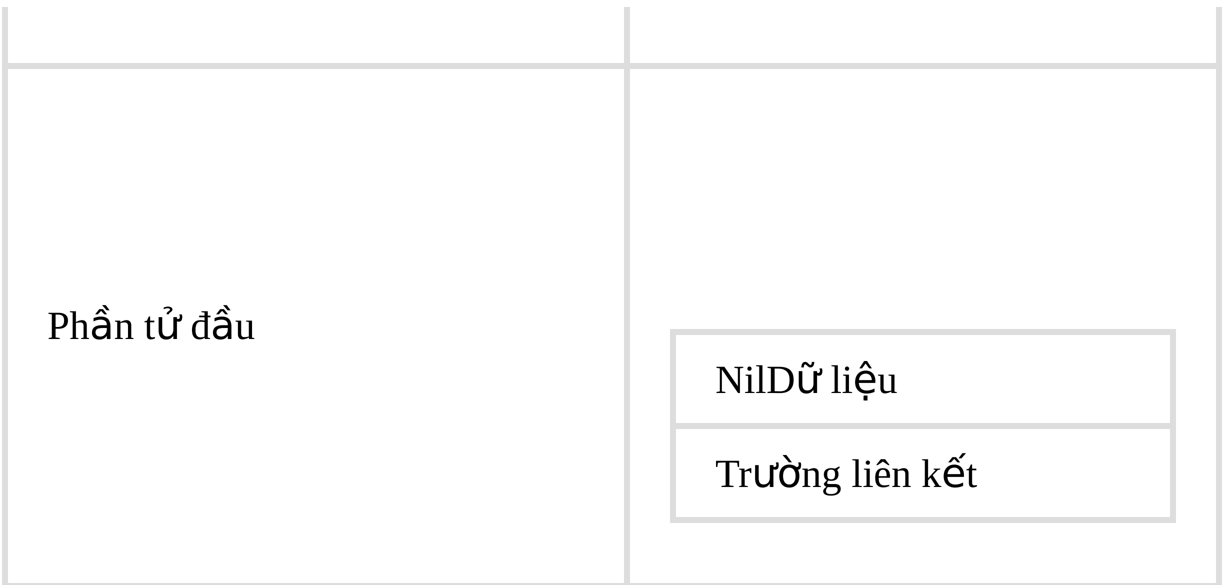
1. Phần tử mới được thêm vào đầu danh sách
2. Phần tử mới được thêm vào cuối danh sách.
3. Phần tử mới được thêm vào một vị trí xác định

Trường hợp a ta có danh sách liên kết ngược (LIFO), còn trường hợp b ta có danh sách liên kết thuận (FIFO) hay còn gọi là hàng đợi QUEUE.

- Danh sách liên kết ngược:

Là loại danh sách mà trường liên kết của phần tử tạo ra sau luôn trỏ vào phần tử trước đó.

Phần tử cuối	Dữ liệu Trường liên kết
Các phần tử trung gian	Dữ liệu Trường liên kết



Type

Ds=^nguoi;

Nguoi=record

Mhs:byte;

Hoten:string[20];

Diem:real;

Tiep:ds;

End;

Sau khi khai báo kiểu dữ liệu cần khai báo biến con trỏ Dslop để lưu trữ dữ liệu nhập vào và biến Ctcuoi (con trỏ cuối) để trỏ vào phần tử cuối cùng.

Đoạn chương trình mô phỏng tạo danh sách liên kết ngược.

Ctcuoi:=nil; {khởi tạo danh sách};

Bắt đầu lặp:

New (dslop); {tạo biến động lưu trữ dữ liệu nhập vào}

Nhập dữ liệu; {nhập dữ liệu cho phần tử thứ i}

Tiep:=ctcuoi; {trường liên kết của phần tử thứ i trỏ vào địa chỉ của con trỏ cuố ctcuoi}

Ctcuoi:=dslop; {hướng ctcuoi vào bản ghi hiện thời}

Kết thúc lặp.

Ví dụ 4.6 : Xây dựng danh sách, chương trình con Hien_LIFO cho hiện dữ liệu lên màn hình theo chiều ngược, phần tử nhập sau hiện trước.

Program dslk_nguoc;

Uses Crt;

Type ds=^nguoi;

Nguoi=record

Mhs:byte;

Hoten:string[20];

Diem:real;

Tiep:ds;

End;

VAr

dslop,ctcuoi:ds; i,j:byte;

tt:char;

Procedure Hien_LIFO;

```
var ct1:ds;

Begin

clrscr;

writeln('Du lieu da nhap-Hien tu cuoi ve dau');

writeln;

ct1:=ctcuoi;

while ct1<>nil do

with ct1^ do

Begin

write(Mhs,' ',hoten);

for j:=1 to (20-length(hoten)) do write('');

writeln(diem:5:2);

ct1:=tiep;

end;

end;

Begin

clrscr;

ctcuoi:=nil; i:=0;

Repeat;

New(dslop); i:=i+1;
```

With dslop^ do

Begin

writeln('ma ho so:',i);mhs:=i;

write('Ho va ten:'); readln(hoten);

write('diem');readln(diem);

tiep:=ctcuoi;

ctcuoi:=dslop;

writeln('nhap tiep khong?C/K'); tt:=readkey;

writeln;

end;

Until tt in ['k','K'];

Hien_lifo;

Readln;

END.

- Hàng đợi Queue-Danh sách liên kết thuận:

Là loại danh sách mà phần tử nào nhập trước thì được lấy ra trước.

Ctdau:=nil; {khởi tạo danh sách};

Bắt đầu lặp:

New (dslop); {tạo biến động lưu trữ dữ liệu nhập vào}

Nhập dữ liệu; {nhập dữ liệu cho phần tử thứ i}

Nếu ctdau=Nil thì ctdau:=dslop;

Ngược lại ctcuoi^.tiep:=dslop;

Ctcuoi:=dslop;

Ctcuoi^.Tiep:=nil; Ctcuoi:=dslop; {hướng ctcuoi vào bản ghi hiện thời}

Kết thúc lặp.

- Chèn thêm phần tử vào danh sách:

Quá trình chèn một phần tử vào danh sách qua các bước sau:

- Xác định vị trí chèn

- Tạo một biến động và xin cấp phát vùng nhớ cho biến động để lưu dữ liệu sẽ chèn vào danh sách.

- Chuyển trường Tiep của phần tử hiện thời đến phần tử bổ sung

- Chuyển trường Tiep của phần tử bổ sung đến phần tử trước phần tử hiện thời.

New(ct1);

With ct1^ do Nhập dữ liệu cho phần tử bổ sung

Dslop:=ctcuoi; {hướng con trỏ đến phần tử cuối cùng trong danh sách}

While (dslop<>nil) and (dslop^.mhs<>n) do

Dslop:=dslop^.tiep; {hướng con trỏ đến vị trí cần chèn}

Ct1^.tiep:=dslop^.tiep;

Dslop^.tiep:=ct1;

Đệ quy